

Deep Learning

15/1/2019

Implementing a Chatbot

Μυλωνάκη Αγγελική

Introduction

In this assignment we were asked to implement a Neural Network based Chatbot (I prefer this term instead of chatbox 🤖). Searching online, one can find multiple solutions on this issue. In this report/implementation the code from [this repository](#) is used and expanded.

High Level Description

When using a chatbot, we expect to be able to ask questions and get suitable answers. This means that the response produced by the bot, needs to somehow relate to the question asked. This is a standard sequence to sequence problem, where the output of the Deep Network relates not only to one input occurrence, but history inputs as well.

In order to implement such a model, a specific architecture is commonly used. This architecture contains an Encoder and a Decoder implemented using Recurrent Neural Networks (RNNs). What the encoder does is take the input data (our question) and train. While training, the state of the RNN cell changes and in the final step the state is passed to the decoder.

In this solution Multiple bidirectional LSTMs are used in combination to Dropout Layers and attention mechanism.

Implementation Details

When sequentially handling data, simple feedforward or convolutional networks perform poorly compared to RNNs, since they struggle reasoning about previous events. RNNs come to the rescue for these kinds of problems by containing loops which allow information to persist. The RNN layers used in this assignment are constituted of:

LSTMs

When it comes to large sequential input, vanilla RNNs fail to capture context information other than the recent. In cases, where we aim to capture long-term dependencies, LSTMs are the preferred approach. In order to capture context, LSTMs remove or add information to the cell state using gates containing a sigmoid and a multiplication function. Given an input, a gate decides how important it is by assigning it a value between zero and one. As a result, the input affects the cell's state based on its significance.

ATTENTION

Attention mechanism is a further enhancement to the Encoder Decoder approach, that enables the RNN to attend to different parts of the source sentence at each time step. As a result, instead of producing a single fixed context vector when given an input, the model learns how to generate a context vector. In other words, we are deciding what is important based on the input and what has been produced so far.

BIDIRECTIONALITY

In order to improve performance, a common approach is to learn representations from both past and future time steps to generate more accurate context and eliminate ambiguity. Similarly, in our implementation, we add a dynamic bidirectional recurrent neural network, taking input and building forward and backward RNNs.

DROPOUT

In order to avoid overfitting, dropout technique is used. By adding this functionality, random neurons are ignored during the training phase and as a result are not considered during a particular forward or backward pass.

OPTIMIZATION

In this implementation, weighted softmax cross entropy loss function is calculated and used for training, along with Adam optimizer. In addition to that gradient clipping technique is added so as to keep our gradients in a reasonable range. Using this approach we minimize the probability of facing exploding/vanishing gradient problems.

Encoder - Decoder


Combining the aforementioned layers and techniques, we create our Encoder and Decoders.

ENCODER

Before feeding the training input (in our case movie dialogs) to the RNN, we preprocess it and learn an embedding layer. This means that we map each word to a low-dimensional, continuous vector. This is then passed to the Encoder for training, causing it to update its cell's state. After that, the last state is passed to the decoder.

DECODER

Two decoders are needed for this task. They share the same architecture and parameters, but have different strategy on how they are fed. The first one is used in the training phase and is



given input, label pairs. These pairs, along with the prediction result of the model are used for back propagation, enabling our model to learn. The second one is the one used for inference (predictions). Since in this case the target sequence is not available, the input of this RNN is solely the output of the previous step.

Embeddings are still needed in this case and are common for both decoders. The main difference is that in the inferencing state, only the embedding parameters are used and not the input.

User interaction

In the notebook handed with this report, you can find 100 chatbot interactions and their result.

Notes and Conclusion

Details regarding the steps of the implementation can be found in the notebook. We need to note the following:

1. A subset of the movie dialog dataset is used for to generate the results
2. When training with the whole dataset it took almost 16 hours for 6 epochs
3. The initial code has been enhanced to store the session after each question and thus avoiding re-training for each question
4. The hyperparameters were tuned based on the produced results and duration of training